리팩토링 할 타이밍

왜?

controller, store, dispather, view...
각각코드는 현재 짧지만 역할이 다름.
유지보수를 위해 별도 파일로 분리하고, 의존성을 정리하면 좋겠음.
의존성은 import ! export !를 사용! (웹팩이 우릴 돕는다)

이런식으로 동작하길 바라며...

```
import BlogListView from './view/bloglistview.js';
import LikeListView from './view/likelistview.js';
import Store from './store.js';
import {Dispatcher} from './lib/util';


export default function Controller(initData) {
    const dispatcher  = new Dispatcher();
    const store       = new Store(dispatcher);
    ....
    ........
```

webpack부터 설치

```
npm init -y
sudo npm install webpack --save-dev
```

index.html과 index.js 만들기

index.html은 javascript 애플리케이션을 만들기 위한 최상위 index역할.

```
//index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>MY Project</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

index.js 는 모든 javascript 의존성의 root역할

```
const root = document.querySelector('#root');
root.innerHTML = `<p>Hello codesquad</p>`;
```

webpack config

참고 : https://webpack.js.org/guides/getting-started/

```
var path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

dev-server를 띄을 수 있는 plugin설치.

webpack-dev-server

```
sudo npm install webpack-dev-server --save-dev
```

webpack 빌드실행해서 dist에 결과 만들기

```
./node_modules/.bin/webpack --config webpack.config.js
```

package.json에 추가하기

```
"scripts": {
    "start": "webpack-dev-server --inline"
  },
```

다시 빌드

```
npm run start
```

서버띄우고 확인.

ex) http://127.0.0.1:8080/

html 꾸미기

기존 html내용을 index.html로 옮기기.

기존 html 삭제.

css를 import문으로 사용하기.

webpack loader에 css loader를 설치/설정해야 한다.

```
import './index.css';
```

```
npm install --save-dev css-loader style-loader
```

css를 위한 webpack config 추가.

module부분을 아래처럼 추가.

```
module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  },
  module: {
      rules: [{
          test: /\.css$/,
          use: [ 'style-loader', 'css-loader' ]
      }]
  }
};
```

css loader**확인**

브라우저에서 css적용된 이쁜 결과를 확인.

style 태그에 css 내용이 옮겨졌음을 확인할 수 있음.

> 참고. css를 별도 css bundle 파일로 만들고 싶다면, 아래 참고해서 웹팩설정 수정.
> https://webpack.js.org/guides/code-splitting-css/#using-extracttextwebpackplugin

그외에도 웹팩에 유용한 loader(module)와 plugin이 잔뜩 있음.

파일 분리

이제 js를 여러개 모듈 단위로 분리.

디렉토리 구조가 이렇게.

```
src
- css
  - index.css
- lib
  - util.js(dispatcher 코드)
- view
  - blogpostview.js
  - likelistview.js
- index.js
- store.js
- controller.js
- index.html
```

의존성 모듈 로딩

import, export를 사용해서 모듈간 의존성 정리.

현재 모듈의 의존성은 어떻게 되어 있는가, 각자 코드 살펴보기?

> index -> controller -> views, store, dispatcher

controller에서 이것저거서 다 사용하고,
controller에서는 dispatcher를 필요한 view에 전달해줄 수도 있고, store의 특정 값을 view에 전달해줄 수도 있음.

index.js example code

```
//index.js example
import './css/index.css';
import Controller from './controller';

const initData = {
  url : "https://tlhm20eugk.execute-api.ap-northeast-2.ama
}

Controller(initData);
```

controller.js example code

```
import BlogListView from './view/bloglistview.js';
import LikeListView from './view/likelistview.js';
import Store from './store.js';
import {Dispatcher} from './lib/util';


export default function Controller(initData) {
        //dispacher를 view와 store에 전달.
    const dispatcher  = new Dispatcher();
    const store       = new Store(dispatcher);
    this.listView     = LikeListView(dispatcher);
    this.blogListView = new BlogListView(dispatcher, initl

    dispatcher.register({
      "CLICK_LIKE" : function(className, str, target) {
        store.changeData(className, str);
        this.blogListView.toggleLikeView(target, classNam
      }.bind(this),
      "INSERT_POSTS" : function(posts) {
        this.blogListView.insertPosts(posts);
      }.bind(this),
      ...
    })
}
```

코드구현 및 리뷰

각자의 상태에서 코드를 구현하고 리뷰해보기.

참고. 웹팩 가이드

가이드를 참고해서 다양한 기능 추가.

https://webpack.js.org/guides/

호환성

es2015코드를 babel-loader를 사용해서 하위 브라우저 지원 권장.

참고 : https://github.com/babel/babel-loader

```
npm i --save-dev babel-loader babel-core
npm i --save-dev babel-preset-env
```

```
module: {
  rules: [
    {
      test: /\.js$/,
      exclude: /(node_modules|bower_components)/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['env']
        }
      }
    }
  ]
}
```

참고. private 변수나 함수를 만들기.

Module pattern.

```
var myModel = (function() {

  const _join = function(name) {return _data + name};
  const _data = "hello, ";

  const modelObj = {
    getPrivateData : function() {return _data},
    getName : function() {return this.name},
    setName : function(name) {
      this.name = _join(name);
    }
  };

  const model = Object.create(modelObj)

  return model;

})();
```